Final Report:

A Computational Logic for Applicative Common Lisp

Contract Number N00014-91-C-0130

0

Final Report:

A Computational Logic for Applicative Common Lisp

Contract Number N00014-91-C-0130

Bob Boyer
Matt Kaufmann
J Moore

Computational Logic Inc.
1717 W. 6th St. Suite 290
Austin, Texas 78703
(512) 322-9951

Table of Contents

This final report is provided to satisfy a contract deliverable. We summarize here our results in automated reasoning under this contract. Many of these words are taken from past reports to ONR.

The development of the Boyer-Moore "Nqthm" theorem prover was supported in part for approximately 20 years by the Office of Naval Research. The long range objective of this research has been to enable programmers to produce software that is mathematically proven to meet its specifications by using mechanical theorem-proving programs that check proofs.

We begin this report with a section listing some accomplishments under this contract related to Nqthm. In particular, the evolution of that system culminated in the release in January, 1994, of the final versions of Nqthm and Pc-Nqthm; see Subsection 1.2 below. Included are results completed during this contract whose work was supported during our preceding ONR contract.

The second section pertains to the main focus of this contract: A Computational Logic for Applicative Common Lisp, or Acl2, which is a theorem proving system that is the successor to Nqthm. The section begins with some background on Acl2, with words describing the Acl2 project from a time shortly before the beginning of this contract. Much progress has been made since that time, which we summarize in the second section of this report, deferring many more details to an Appendix.

The third section lists publications, reports, presentations, and awards and honors, as drawn from previous ONR annual reports. We conclude with the Appendix mentioned above.

1. Results outside Acl2

1.1 UNITY

Some papers by David Goldschlag on his mechanization of the UNITY logic using Nqthm appeared during this contract. They are listed in Section 3.

1.2 Nqthm Release

In January, 1994, We completed the final released of Nqthm and Pc-Nqthm, which are generally known as "Boyer-Moore theorem prover" and "its interactive enhancement." The systems are publicly available from Internet host ftp.cli.com and include many megabytes of proved theorems.

The final release of Nqthm cleans up a few details and, perhaps more importantly, includes an assemblage of *many thousands* of theorems successfully processed by the system; similarly for Pc-Nqthm. This is a quantitative accomplishment for which ONR

can be proud (and has received much credit). It seems clear that this is the largest collection of formal mathematics ever assembled. We believe that this massive collection of theorems provides strong evidence that existing technology can be used to proof check mechanically essentially anything that can be proved by hand.

As an example, the so-called "CLInc stack" of computing systems, which was the focus of an issue of the *Journal of Automated Reasoning*, contains the proof of correctness of a compiler from a small Pascal-like language; a proof of correctness of an assembler from that compiler's target down to a machine language, and a proof of correctness of an implementation of that machine language in hardware. A successor to that machine has, in fact, been fabricated, and the entire "stack" proof has been ported to sit on that processor. Thus, Nqthm and Pc-Nqthm have been used to prove the correctness of a high-level language on a fabricated processor, which may be the single greatest accomplishment in mechanized formal methods.

Computational Logic's Technical Report 75, available upon request, summarizes these proofs using Nqthm and Pc-Nqthm. We should also mention that we have brought the Nqthm documentation up to date.

1.3 Mechanizing Quantification

With ONR support in previous years, we have *created an implementation* of full first-order quantification for Nqthm. That facility is part of Pc-Nqthm.

In Computational Logic Technical Report 81, we present an implementation of a *recognizer* for quantified notions defined in Nqthm. That is, we provide a method for checking that a given function (defined without quantifiers) does indeed represent a quantified notion. We also present methods for *generating* constructively-presented functions that represent quantified notions, including definitions using only bounded quantifiers.

We have begun investigation general "binding" mechanisms, as described in our 1992 annual report to ONR, but have nothing further to report at this time.

1.4 An Application

We have used Nqthm to formally specify and verify properties of a simple train crossing gate system. This problem has been suggested by Connie Heitmeyer of NRL as a benchmark for evaluating the performance of specification tools and automated reasoning systems in the area of safety-critical systems. The work has been documented in Computational Logic's Technical Report 93.

2. Acl2

Why change from Nqthm to Acl2?

Scale and Performance. Nqthm was designed for smaller problems than we face today. For example, it is difficult for several users to contribute to the same project with Nqthm, because there is no notion of hierarchical structure for ''subproofs.''

Control and Flexibility. Nqthm is hard to control and tailor to the individual user. For example, there is no facility in Nqthm for defining macros.

Practicing What We Preach. Acl2 is programmed essentially in itself, and we believe that by forcing ourselves to use the programming environment that we offer to others and support in our logic, we are creating a really usable programming environment for which there is a powerful verifier. Also, applicative programming holds a promise for high performance via parallel implementations.

Acl2 addresses all of these issues with considerable success.

But now let us back up, giving some perspective on the Acl2 project. The following subsection is excerpted almost exactly from Boyer and Moore's contribution to the Tenth International Conference on Automated Deduction, 1990, where they gave the keynote address. This subsection will be followed by a report on progress on the items listed in the statement of work of this contract. Further progress is alluded to in the final subsection, with details reported in the Appendix.

2.1 Background on Acl2 (essentially from CADE-10 keynote address)

We are currently constructing an entirely new version of our prover. The name of the new system is *A Computational Logic for Applicative Common Lisp*, which might be abbreviated as ''ACL ACL'' but which we abbreviate as ''Acl2.'' Whereas Nqthm has been available for some time, extensively documented, and widely used, Acl2 is still very much under development. Hence the following remarks are somewhat speculative.

Instead of supporting ''Boyer-Moore logic'', which reflects an odd mixture of functions vaguely, but not consistently, related to Lisp 1.5 and Interlisp, Acl2 directly supports perfectly and accurately (we hope) a large subset of applicative Common Lisp. That is, Acl2 is to applicative Common Lisp what Nqthm is to the ''Boyer-Moore logic'', a programming/theorem proving environment for an executable logic of recursive functions.

More precisely, we have identified an applicative subset of Common Lisp and axiomatized it, carefully following Steele's *Common Lisp: The Language*. Because arrays, property lists, input/output and certain other commonly used programming features are not provided applicatively in Common Lisp (i.e., they all involve the notion of explicit state changes), we axiomatized applicative versions of these features. For

example, when one "changes" an array object, one gets a new array object. However, we gave these applicative functions very efficient implementations which are in complete agreement with their axiomatic descriptions but which happen to execute at near von Neumann speeds when used in the normal von Neumann style (in which "old" versions of a modified structure are not accessed). The result is "applicative Common Lisp" which is also an executable mathematical logic.

Like Nqthm, the logic of applicative Common Lisp provides a definitional principle that permits the sound extension of the system via the introduction of recursive functions. Unlike Nqthm, however, functions in applicative Common Lisp may be defined only on a subset of the universe. Like Nqthm, the new logic provides the standard first order rules of inference and induction. However, the axioms are different since, for example, Nqthm and Acl2 differ on what `(CAR NIL)` is. Most importantly for the current purposes, we claim that all correct Common Lisps implement applicative Common Lisp directly and that, unlike Nqthm's logic, applicative Common Lisp is a practical programming language.

Acl2 is a theorem prover and programming/proof environment for applicative Common Lisp. Acl2 includes all of the functionality of Nqthm (as understood in the new setting) plus many new features (e.g., congruence-based rewriting). The source code for Acl2 consists of about 1.5 million characters, all but 43,000 of which are in applicative Common Lisp. That is, 97% of Acl2 is written applicatively in the same logic for which Acl2 proves theorems. The 3% of non-applicative code is entirely at the top-level of the read-eval-print user interface and deals with reading user input, error recovery and interrupts. We expect to implement `read` applicatively and limit the non-applicative part of Acl2 to the essential interaction with the underlying Common Lisp host system.

Thus, in Acl2 as it currently stands, the definitional principle is implemented as a function in logic, including the syntax checkers, error handlers, and data base handlers. The entire "Boyer-Moore theorem prover" -- as that term is now understood to mean "the theorem prover Boyer and Moore have written for Acl2" -- is a function in the logic, including the simplifiers, the decision procedures, the induction heuristics, and all of the proof description generators.

The fact that almost all of Acl2 is written applicatively in the same logic for which it is a theorem prover allows the Acl2 source code to be among the axioms in that definitional extension of the logic. The user of the Acl2 system can define functions, combine his functions with those of Acl2, execute them, or prove things about them, in a unified setting. One need only understand one language, Common Lisp, to use the "logic", interact with the system, interface to the system, or modify the system. DEFMACRO can be used to extend the syntax of the language, users can introduce their own front-ends by programming within the logic, and all of the proof routines are accessible to users and have exceptionally clear (indeed, applicative) interfaces. Many new avenues in metatheoretic extensibility are waiting to be explored. We believe we have taken a major step towards the goal of perhaps someday checking the soundness of most of the theorem prover by defining the theorem prover in a formalized logic.

At the time of this writing, we have completely recoded all of the functionality of Nqthm, but have only begun experimentation with proving theorems. However, our preliminary evidence is that there will be no substantial degradation in performance, even though Acl2 is coded applicatively.

2.2 Acl2 Accomplishments

As mentioned above, the axiomatic definition of Acl2 is consistent with the Common Lisp manuals by Guy Steele. Thus the new logic is directly implemented by Lucid, Allegro, Symbolic, KCL, etc., and Acl2, in turn, directly provides a programming/verification environment for those systems. Most stunning however is the fact that the new system is written in Acl2. That is, it is an applicative program in the logic; this includes the syntax checking, simplifiers, decision procedures, io, error handling, data base maintenance, etc. The Acl2 code is part of the Acl2 logical data base, and hence the system provides exceptional logical coherence, power, extensibility and unity. Early performance experiments indicate that Acl2 is as fast as Nqthm. We believe Acl2 represents a revolutionary step forward in verification, programming languages, and theorem proving. In addition, Acl2 unites several disparate communities, offers an extensive test bed for research into parallelism in an applicative setting, and opens new doors in both theorem proving and metatheoretic extensibility. We have in fact begun to think about using Acl2 to prove itself correct (and have considered ways of avoiding the apparent circularity of such an approach).

Let us turn now to specific items from the Statement of Work. We have made considerable progress on Acl2 since the time of the description relayed in Subsection 2.1 above. We should mention that several contracts at Computational Logic, Inc. have supported our Acl2 work, and we do not attempt to single out which work below was due in particular to ONR support. However, here we organize the progress most relevant to this contract according the corresponding items in its Statement of Work, which specified some directions in which to concentrate our efforts.

- **Verify (parts of) the acl2 code.**
  We have successfully proved termination for many of the definitions that are part of the Acl2 system. We have also proved that many of those functions may be admitted as "`:gold`" functions, i.e., that they are guaranteed to evaluate without error (except possibly for resource errors).

- **Provide a richer mechanism for metatheoretic extensibilty.**
  We have designed and implemented a mechanism for "conditional metalemmas," i.e., metalemmas with meta-level hypotheses.

- **Provide further interactive features and begin verifying their correctness.**
  We have incorporated a rich interactive capability into Acl2, but have not yet begun on its verification.

• **Extend acl2 by including more seemingly non-applicative features.**
At this point we have not seen the need to make such extensions. However, in related work, Bishop Brock of CLI has prove a collection of theorems about Acl2 arrays.

• **Build a much more useful arithmetic reasoning capability.**
We have made technical improvements in the linear arithmetic reasoning code in Acl2, and have also provided a mechanism whereby the user can specify different algebraic systems to use in place of rational number arithmetic. This mechanism has been used to implement a capability for Acl2 to emulate Nqthm. When that capability has been polished it will, in turn, be very useful in testing Acl2, since the Nqthm examples containing perhaps 16,000 definitions and theorems will become usable for testing Acl2. Finally, we should mention that we have accumulated a reasonably large collection of useful, proved arithmetic facts (mostly rewrite rules, but a few others including three metalemmas).

• **Provide first-order capabilities.**
Acl2 has a mature method for introducing constrained functions that can sometimes be used in place of quantification for building mathematical models. If we introduce traditional first-order quantifiers, the work reported above in Subsection 1.3 should be helpful.

• **Create a facility for management of reusable, incremental theories.**
The Acl2 *book* mechanism is fully operational and mature. It allows for the development of partially ordered collections of books. A book is nothing more than a file containing Acl2 *event forms* (most commonly, definitions and theorems). Note that Acl2 supports compilation of books, which is important for efficient execution.

• **Exercise the system and demonstrate its capabilities.**
We have continually exercised the system, both in processing its own definitions and in implementing the Nqthm package. We have also carried out an experiment in non-standard analysis with ONR support that suggests a way to support continuous mathematics in Acl2; we expect to write that up this year. Other projects at CLI include definitions and theorems proved related to interpreters for a DSP chip and for subsets of Ada and C, as well as what one might call a ''verified VCG generator'' for a toy programming language. Finally, we have used the system as an efficient programming environment for a fast OBDD (ordered binary decision diagram) algorithm for deciding propositional logic formulas, and for a 1993 compliant VHDL parser.

2.3 Acl2: Concluding Words

We continue to make progress on Acl2. Although we have summarized the progress areas above that relate most directly to this contract's statement of work, much more has been accomplished. We have included an appendix to this report in order to give some flavor for what is involved in the evolution of Acl2.

3. Publications, Reports, Presentations, Awards/Honors

Below we list publications, reports, presentations, and awards and honors from the annual reports filed under this contract.

3.1 Publications

- Response and biographical sketch in receipt by Boyer and Moore of the 1991 AMS Current Award for Automatic Theorem Proving, in: "Automatic Theorem Proving Awards Presented," Notices of the American Mathematical Society, vol. 38, no. 5, pp. 405-410.

- Bob Boyer, Matt Kaufmann, and J Strother Moore, "The Boyer-Moore Theorem Prover and its Interactive Enhancement," submitted for publication.

- R. S. Boyer and J S. Moore, "MJRTY - A Fast Majority Vote Algorithm," in Robert S. Boyer, editor, Automated Reasoning: Essays in Honor of Woody Bledsoe. Kluwer Academic, Dordrecht, The Netherlands, 1991, pp. 105-117.

- Robert S. Boyer, D. Goldschlag, M. Kaufmann, and J Strother Moore, "Functional Instantiation in First Order Logic," in V. Lifschitz (editor), Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. Academic Press, 1991. pp. 7-26.

- D. Goldschlag, "A Mechanical Formalization of Several Fairness Notions," to appear in proceedings of VDM '91, Amsterdam, October 1991.

- D. Goldschlag, "Mechanically Verifying Safety and Liveness Properties of Delay Insensitive Circuits," in: proceedings of Computer Aided Verification 1991, Aalborg, Denmark, July 1991.

- D. Goldschlag, "Mechanically Verifying Safety and Liveness Properties of Delay Insensitive Circuits," in *Computer Aided Verification*, K. G. Larsen, A. Skou (editors), Springer-Verlag Lecture Notes in Computer Science 575, Berlin, 1992.

- D. Goldschlag, "Verifying Safety and Liveness Properties of a Delay Insensitive FIFO Circuit on the Boyer-Moore Prover," 1991 International Workshop on Formal Methods in VLSI Design, Miami, January 1991.

- M. Kaufmann, "Generalization in the Presence of Free Variables: a

Mechanically-Checked Correctness Proof for One Algorithm,'' J. Automated Reasoning, volume 7, 1991.

• Matt Kaufmann, ''An Extension of the Boyer-Moore Theorem Prover to Support First-Order Quantification,'' J. Automated Reasoning 9, December 1992, pp. 355-372.

3.2 Reports

• David M. Goldschlag, ''Mechanically Verifying Concurrent Programs,'' Technical Report 71, Computational Logic, Inc., September 1991.

• M. Kaufmann, ''An informal discussion of issues in mechanically-assisted reasoning,'' CLI Internal Note 242, September, 1991; to appear in proceedings of 1991 International Workshop on the HOL Theorem Proving System and its Applications, University of California, Davis, August 27-30, 1991.

• Matt Kaufmann, ''A Strategy for "Constructivizing" DEFN-SK,'' Internal Note 250, Computational Logic Inc., December 1991.

• Matt Kaufmann, ''Quantification in Nqthm: a Recognizer and Some Constructive Implementations,'' Technical Report 81, Computational Logic Inc., August 1992.

• William D. Young, ''Verifying a Simple Real-Time System with Nqthm,'' Technical Report 93, Computational Logic, Inc., September 1993.

3.3 Presentations

• ''Mechanized Correctness Proofs of Some M68020 Programs,'' Robert S. Boyer, Symposium in honor of John McCarthy, Stanford, California, September 1991.

• ''A Theorem Prover for a Computational Logic,'' Robert S. Boyer, Research Institute for Mathematical Sciences at the Univ. of Kyoto, Dec. 12, 1990.

• ''A Hardware Reset Lemma and Its Proof,'' Matt Kaufmann, Japanese-American Workshop on Automated Reasoning, Argonne National Labs, June 1991.

• Matt Kaufmann, IRST, Trento, Italy, June 1991 (invited guest for a week).

• ''An informal discussion of issues in mechanically-assisted reasoning,'' August 30, 1991 -- a keynote address at the 1991 International Workshop on the HOL Theorem Proving System and its Applications, University of California, Davis.

• Boyer and Moore, acceptance speech for AMS Prize for Current Achievements in Automatic Reasoning in January, 1991 at the AMS conference in San Francisco.

• "A Mechanical Formalization of Several Fairness Notions," David Goldschlag, VDM '91, Amsterdam, October 1991.

• "Mechanically Verifying Safety and Liveness Properties of Delay Insensitive Circuits," David Goldschlag, Computer Aided Verification 1991, Aalborg, Denmark, July 1991.

• "Verifying Safety and Liveness Properties of a Delay Insensitive FIFO Circuit on the Boyer-Moore Prover," David Goldschlag, 1991 International Workshop on Formal Methods in VLSI Design, Miami, January 1991.

• J Moore, "The Role of Automated Reasoning in Integrated System Verification Environments," Workshop on the Effective Use of Automated Reasoning Technology in System Development, April 6-8, 1992, Naval Research Laboratory.

• Matt Kaufmann, "Should We Begin a Standardization Process for Interface Logics?" Workshop on the Effective Use of Automated Reasoning Technology in System Development, April 6-8, 1992, Naval Research Laboratory.

• J Moore, "The Formal Specification of Programs that Interact with Hostile Environments," NSA, July 1992.

• J Moore, "The Acl2 Project," NSA, July 1992.

• J Moore, presentation on the paper "A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol," August, 1992, NASA Langley.

• Bob Boyer and Yuan Yu, "Automated Correctness Proofs of Machine Code Programs for a Commercial Microprocessor," CADE-11, Saratoga Springs, NY, June 1992.

• Bob Boyer, invited opening talk at the Workshop on Automation of Induction, Saratoga Springs, June 1992.

• Matt Kaufmann, talk at NSA on Nqthm and Pc-Nqthm, April 1992.

• Bob Boyer, Distinguished Lecturer, Harvard University, December 2, 1992.

• Bob Boyer, Distinguished Lecturer, University of Illinois, Urbana, Illinois, April 26, 1993.

• Bob, Boyer, Dagstuhl (Germany) Seminar on Automated Deduction, March 9, 1993.

• Bob Boyer, Annual Meeting of the Esprit Basic Research Action on Proofs and Types, Nijmegen, the Netherlands, May 25, 1993.

• Bob Boyer, German Institute for Artificial Intelligence (DFKI), University of Saarbruecken, Germany, June 2, 1993.

• Matt Kaufmann, "Interactive Proving Using the Boyer-Moore Theorem

Provers." *Formal Methods in Software Engineering: AUTOMATED REASONING*, Workshop sponsored by ARO and ONR, University of Pennsylvania, Philadelphia, Pennsylvania, May 10--11, 1993.

• J Moore, "Mechanized Logic and Mathematical Modeling of Digital Systems." Boeing Formal Methods Lecture Series Seattle, Washington, May 3, 1993.

• J Moore, "An Nqthm Tutorial." Boeing Formal Methods Lecture Series Seattle, Washington, May 3, 1993.

• J Moore, "Mathematical Modeling of Digital Systems." Digital Equipment Corporation Systems Research Center, Palo Alto, California, May 4, 1993.

• J Moore, "Packages in Acl2." Digital Equipment Corporation Systems Research Center, Palo Alto, California, May 4, 1993.

3.4 Awards/honors

The American Mathematical Society awarded Boyer and Moore their Prize for Current Achievements in Automatic Reasoning in January, 1991, at the AMS conference in San Francisco. This is a significant recognition of the collaboration by Boyer and Moore that has been continuing since the early 70s, with substantial ongoing support by ONR. Pages 407-410 of the Notices of the American Mathematical Society, vol. 38, no. 5, contains descriptions of NQTHM and its applications, plus biographical remarks by Boyer and Moore about the development of their system and thanks to ONR and other research sponsors, all in acknowledgement of receipt of this prize.

Bob Boyer was a member of Organizing Board and Program Committee of 1993 Workshop on Automated Induction, held in conjunction with AAAI, July, 1993.

Bob Boyer also became a member of the Organizing Committee for 1995 Dagstuhl (Germany) Workshop on Automated Induction.

Matt Kaufmann became a member of the CADE-12 program committee, and co-organized a workshop at that conference.

Appendix A
Enhancements to Acl2

Here is a list of the improvements and changes made to Acl2, as advertised in its various (internal) release notes.

VERSION 1.2 RELEASE NOTES

```
Hacker mode has been eliminated and programming mode has been added.
Programming mode is unsound but does syntax checking and permits
redefinitions of names.  See :doc load-mode and :doc g-mode.

The arguments to LD have changed.  LD is now much more sophisticated.
See :DOC ld.

For those occasions on which you wish to look at a large list structure
that you are afraid to print, try (walkabout x state), where x is
an Acl2 expression that evaluates to the structure in question.  I
am afraid there is no documentation yet, but it is similar in spirit
to the Interlisp structure editor.  You are standing on an object and
commands move you around in it.  E.g., 1 moves you to its first element,
2 to its second, etc.; 0 moves you up to its parent; nx and bk move
you to its next sibling and previous sibling; pp prettyprints it;
q exits returning nil; = exits returning the thing you're standing on;
(= symb) assigns the thing you're standing on to the state global
variable symb.

Several new hints have been implemented, including :by and :do-not.
The old :do-not-generalize has been scrapped in favor of such new
hints as :do-not (generalize elim).  :By lets you say ''this goal is
subsumed by'' a given lemma instance.  The :by hint also lets you
say ''this goal can't be proved yet but skip it and see how the rest
of the proof goes.'' See :DOC hints.
```

VERSION 1.3 RELEASE NOTES

Programming mode has been eliminated. Instead, all functions have a ''color'' which indicates what can be done with the function. For example, :red functions can be executed but have no axioms describing them. Thus, :red functions can be introduced after passing a simple syntactic check and they can be redefined without undoing. But nothing of consequence can be proved about them. At the other extreme are :gold functions which can be executed and which also have passed both the termination and the guard verification proofs. The color of a function can be specified with the new XARGS keyword, :COLOR, which, if omitted defaults to the global setting of LD-COLOR. LD-COLOR replaces LOAD-MODE. Setting LD-COLOR to :red causes behavior similar to the old :g-mode. Setting LD-COLOR to :gold causes behavior similar to the old :v-mode. It is possible to prototype your system in :red and then convert :red functions to :blue individually by calling verify-termination on them. They can then be converted to :gold with verify-guards. This allows us to undertake to verify the termination and guards of system functions. See :DOC color for an introduction to the use of colors.

Type prescription rules have been added. Recall that in Nqthm. some REWRITE rules were actually stored as ''type-prescriptions.'' Such rules allow the user to inform Nqthm's primitive type mechanism as to the kinds of shells returned by a function. Earlier versions of Acl2 did not have an analogous kind of rule because Acl2's type mechanism is complicated by guards. Version 1.3 supports TYPE-PRESCRIPTION rules. See :DOC type-prescription.

Three more new rule-classes implement congruence-based rewriting. It is possible to identify a binary relation as an equivalence relation (see :DOC equivalence), to show that one equivalence relation refines another (see :DOC refinement) and to show that a given equivalence relation is maintained when rewriting a given function call, e.g., (fn ...xk...), by maintaining another equivalence relation while rewriting the kth argument (see :DOC congruence). If r has been shown to be an equivalence relation and then (implies hyps (r (foo x) (bar x))) is proved as a :REWRITE rule, then instances of (foo x) will be replaced by corresponding instances of (bar x) provided the instance occurs in a slot where the maintainence of r-equivalence is known to be sufficient and hyps can be established as usual.

In Version 1.2, rule-classes were simple keywords, e.g., :REWRITE or :ELIM. In Version 1.3, rule-classes have been elaborated to allow you to specify how the theorem ought to be used as a rule. That is, the new rule-classes allows you to separate the mathematical statement of the formula from its interpretation as a rule. See :DOC rule-classes.

Rules used to be named by symbols, e.g., CAR and CAR-CONS were the names of rules. Unfortunately, this was ambiguous because there are three rules associated with function symbols: the symbolic definition, the executable counterpart, and the type-prescription; many different rules might be associated with theorems, depending on the rule classes. In Version 1.3 rules are named by ''runes'' (which is just short hand for ''rule names''). Example runes are (:DEFINITION CAR), (:EXECUTABLE-COUNTERPART CAR), and (:TYPE-PRESCRIPTION CAR . 1). Every rule added by an event has a different name and you can enable

and disable them independently. See :DOC rune and :DOC theories.

The identity function FORCE, of one argument, has been added and given a special interpretation by the functions responsible for establishing hypotheses in backchaining: When the system fails to establish some hypothesis of the form (FORCE term), it simply assumes it is true and goes on, delaying until later the establishment of term. In particular, pushes a new subgoal to prove term in the current context. When that subgoal is attacked, all of the resources of the theorem prover, not just rewriting, are brought to bear. Thus, for example, if you wish to prove the rule

```
 (IMPLIES (GOOD-STATEP s) (EQUAL (EXEC s n) s'))
```

and it is your expectation that every time EXEC appears its first argument is a GOOD-STATEP then you might write the rule as

```
 (IMPLIES (FORCE (GOOD-STATEP s)) (EQUAL (EXEC s n) s')).
```

This rule is essentially an unconditional rewrite of (EXEC s n) to s' that spawns the new goal (GOOD-STATEP s). See :DOC force. Because you can now specify independently how a theorem is used as a rule, you need not write the FORCE in the actual theorem proved. See :DOC rule-classes.

Version 1.3 supports a facility similar to Nqthm's BREAK-LEMMA. See :DOC break-rewrite. You can install ''monitors'' on runes that will cause interactive breaks under certain conditions.

Acl2 also provides ''wormholes'' which allow you to write functions that cause interaction with the user but which do not require that you have access to STATE. See :DOC wormhole.

The rewriter now automatically backchains to stronger recognizers. There is no user hook to this feature but it may simplify some proofs with which older versions of Acl2 had trouble. For example, if the rewriter is trying to prove (rationalp (foo a b c)) it is now smart enough to try lemmas that match with (integerp (foo a b c)).

VERSION 1.4 RELEASE NOTES

Once again LD only takes one required argument, as the bind-flg has been deleted.

Three commands have been added in the spirit of :PE. :PE! is similar to :PE but it prints all events with the given name, rather than just the most recent. The command :PF prints the corollary formula corresponding to a name or rune. The command :PL (print lemmas) prints rules whose top function symbol is the given name. See :DOC pe!, :DOC pf, and :DOC pl.

Book naming conventions have been changed somewhat. The once-required .lisp extension is now prohibited! Directories are supported, including a notion of ``connected book directory''. See :DOC book-name. Also, the second argument of certify-book is now optional, defaulting to 0.

Compilation is now supported inside the Acl2 loop. See :DOC comp and :DOC set-compile-fns.

The default color is now part of the Acl2 world; see :DOC default-color. Ld-color is no longer an LD special. Instead, colors are events; see :DOC red, :DOC pink, :DOC blue, and :DOC gold.

A table exists for controlling whether Acl2 prints comments when it forces hypotheses of rules; see :DOC force-table. Also, it is now possible to turn off the forcing of assumptions by disabling the definition of force; see :DOC force.

The event defconstant is no longer supported, but a very similar event, defconst, has been provided in its place. See :DOC defconst.

The event for defining congruence relations is now defcong (formerly, defcon).

Patterns are now allowed in :expand hints. See the documentation for :expand in :DOC hints.

We have improved the way we report rules used by the simplifier. All runes of the same type are reported together in the running commentary associated with each goal, so that for example, executable counterparts are listed separately from definitions, and rewrite rules are listed separately from linear rules. The preprocessor now mentions ``simple'' rules; see :DOC simple.

The mechanism for printing warning messages for new rewrite rules, related to subsumption, now avoids worrying about nonrecursive function symbols when those symbols are disabled. These messages have also been eliminated for the case where the old rule is a :definition rule.

Backquote has been modified so that it can usually provide predictable results when used on the left side of a rewrite rule.

Time statistics are now printed even when an event fails.

The Acl2 trace package has been modified so that it prints using the values of the Lisp globals *print-level* and *print-length* (respectively).

Table has been modified so that the :clear option lets you replace the entire table with one that satisfies the val and key guards (if any); see :DOC table.

We have relaxed the translation rules for :measure hints to defun, so that the the same rules apply to these terms that apply to terms in defthm events. In particular, in :measure hints mv is treated just like list, and state receives no special handling.

The loop-stopper test has been relaxed. The old test required that every new argument be strictly less than the corresponding old argument in a certain term-order. The new test uses a lexicographic order on term lists instead. For example, consider the following rewrite rule.

```
  (equal
   (variable-update var1
                    val1 (variable-update var2 val2 vs))
   (variable-update var2
                    val2 (variable-update var1 val1 vs)))
```

This rule is permutative. Now imagine that we want to apply this rule to the term

```
  (variable-update u y (variable-update u x vs)).
```

Since the actual corresponding to both var1 and var2 is u, which is not strictly less than itself in the term-order, this rule would fail to be applied in this situation when using the old test. However, since the pair (u x) is lexicographically less than the pair (u y) with respect to our term-order, the rule is in fact applied using our new test.

Messages about events now contain a space after certain left parentheses, in order to assist emacs users. For example, the event

```
  (defthm abc (equal (+ (len x) 0) (len x)))
```

leads to a summary containing the line

```
  Form:  ( DEFTHM ABC ...)
```

and hence, if you search backwards for ``(defthm abc'', you won't stop at this message.

More tautology checking is done during a proof; in fact, no goal printed to the screen, except for the results of applying :USE and :BY hints or the top-level goals from an induction proof, are known to Acl2 to be tautologies.

The ld-query-control-alist may now be used to suppress printing of queries; see :DOC ld-query-control-alist.

Warning messages are printed with short summary strings, for example the string ''Use'' in the following message.

```
Acl2 Warning [Use] in DEFTHM:  It is unusual to :USE an enabled :REWRITE
or :DEFINITION rule, so you may want to consider disabling FOO.
```

At the end of the event, just before the time is printed, all such summary strings are printed out.

The keyword command :u has been introduced as an abbreviation for :ubt :max. Printing of query messages is suppressed by :u.

The keyword :cheat is no longer supported by any event form.

Some irrelevant formals are detected; see :DOC irrelevant-formals.

A bug in the application of metafunctions was fixed: now if the output of a metafunction is equal to its input, the application of the metafunction is deemed unsuccessful and the next metafunction is tried.

An example has been added to :DOC equivalence to suggest how to make use of equivalence relations in rewriting.

The following Common Lisp functions have been added to Acl2: alpha-char-p, upper-case-p, lower-case-p, char-upcase, char-downcase, string-downcase, string-upcase, and digit-charp-p.

A documentation section called Proof-checker has been added for the interactive facility, whose documentation has been slightly improved. See in particular :DOC proof-checker, :DOC verify, and :DOC macro-command.

A number of events that had been inadvertently disallowed in books are now permitted in books. These are: defcong, defcor, defequiv, defrefinement, defstub, and verify-termination.

VERSION 1.5 RELEASE NOTES

Acl2 now allows ''complex rationals,'' which are complex numbers whose real parts are rationals and whose imaginary parts are non-zero rationals. See :DOC complex.

A new way of handling FORCEd hypotheses has been implemented. Rather than cause a case split at the time the FORCE occurs, we complete the main proof and then embark on one or more ''forcing rounds'' in which we try to prove the forced hypotheses. See :DOC forcing-round.

To allow us to compare the new handling of FORCE with the old, Version 1.5 implements both and uses a flag in STATE to determine which method should be used. Do (assign old-style-forcing t) if you want FORCE to be handled as it was in Version 1.4. However, we expect to eliminate the old-style forcing eventually because we think the new style is more effective.

To see the difference between the two approaches to forcing, try proving the associativity of append under both settings of old-style-forcing. To get the new behavior invoke:

```
(thm (implies (and (true-listp a) (true-listp b))
              (equal (append (append a b) c)
                     (append a (append b c)))))
```

Then (assign old-style-forcing t) and invoke the thm command above again.

A new :cases hints allows proof by cases. See :DOC hints.

Include-book and encapsulate now restore the acl2-defaults-table when they complete. See :DOC include-book and :DOC encapsulate.

The guards on many Acl2 primitives defined in axioms.lisp have been weakened to permit them to be used in accordance with lisp custom and tradition.

It is possible to attach heuristic filters to :REWRITE rules to limit their applicability. See :DOC syntaxp.

A tutorial has been added; see :DOC tutorial.

Events now print the Summary paragraph listing runes used, time, etc., whether they succeed or fail. The format of the ''failure banner'' has been changed but still has multiple asterisks in it. THM also prints a Summary, whether it succeeds or fails; but THM is not an event.

A new event form skip-proofs has been added; see :DOC skip-proofs.

A user-specific customization facility has been added in the form of a book that is automatically included, if it exists on the current directory. See :DOC acl2-customization.

A facility for conditional metalemmas has been implemented; see :DOC meta.

The acceptable values for ld-skip-proofsp have changed. In the old version (Version 1.4), a value of t meant that proofs and LOCAL events are to be skipped. In Version 1.5, a value of t means proofs (but not LOCAL events) are to be skipped. A value of 'include-book means proofs and LOCAL events are to be skipped. There are two other, more obscure, acceptable values. See :DOC ld-skip-proofsp.

```
In order to turn off the forcing of assumptions, one should now
disable the :executable-counterpart of force (rather than the
:definition of force, as in the previous release); see :DOC force.

The macros enable-forcing and disable-forcing make it convenient to enable
or disable forcing. See :DOC enable-forcing and :DOC disable-forcing.

The new commands :pr and :pr! print the rules created by an event or
command. See :DOC pr and :DOC pr!.

The new history commands :puff and :puff* will replace a compound command
such as an encapsulate or include-book by the sequence of events in it. That
is, they ''puff up'' or ''lift'' the subevents of a command to the command
level, eliminating the formerly superior command and lengthening the history.
This is useful if you want to ''partially undo'' an encapsulate or book or
other compound command so you can experiment. See :DOC puff and see
:DOC puff*.

Theory expressions now are allowed to use the free variable WORLD and
prohibited from using the free variable STATE. See :DOC theories, although
it is essentially the same as before except it mentions WORLD instead of STATE.
See :DOC world for a discussion of the Acl2 logical world. Allowing
in-theory events to be state-sensitive violated an important invariant about
how books behaved.

TABLE keys and values now are allowed to use the free variable WORLD
and prohibited from using the free variable STATE. See the note above
about theory expressions for some explanation.

The macro for minus, -, used to expand (- x 3) to (+ x -3) and now expands it
to (+ -3 x) instead. The old macro, if used in the left-hand sides of
rewrite rules, produced inapplicable rules because the constant occurs in the
second argument of the +, but potential target terms generally had the
constant in the first argument position because of the effect of
commutativity-of-+.

A new class of rule, :Linear-alias rules, allows one to implement
the nqthm package and similar hacks in which a disabled function is
to be known equivalent to an arithmetic function. See :DOC linear-alias.

A new class of rule, :BUILT-IN-CLAUSE rules, allows one to extend the set of
clauses proved silently by DEFUN during measure and guard processing. See
:DOC built-in-clauses.

The new command PCB! is like PCB but sketches the command and then prints its
subsidiary events in full. See :DOC pcb!.

:REWRITE class rules may now specify the :LOOP-STOPPER field. See
:DOC rule-classes and :DOC loop-stopper.

The rules for how loop-stoppers control permutative rewrite rules
have been changed. One effect of this change is that now when the
built-in commutativity rules for + are used, the terms a and (- a)
are permuted into adjacency. For example, (+ A B (- A)) is now
normalized by the commutativity rules to (+ A (- A) B); in Version
1.4, B was considered syntactically smaller than (- A) and so (+ A B
(- A)) is considered to be in normal form. Now it is possible to
```

```
arrange for unary functions be be considered ''invisible'' when they
are used in certain contexts.  By default, unary-- is considered
invisible when its application appears in the argument list of
binary-+.  See :DOC loop-stopper and :DOC set-invisible-fns-alist.

Extensive documentation has been provided on the topic of Acl2's
''term ordering.''  See :DOC term-order.

Calls of LD now default ld-error-action to :RETURN rather than to the current
setting.

The command descriptor :x has been introduced and is synonymous with
:max, the most recently executed command.  History commands such as
:pbt print a :x beside the most recent command, simply to indicate that
it IS the most recent one.

The command descriptor :x-23 is synonymous with (:X -23).  More generally,
every symbol in the keyword package whose first character is #\\X and whose
remaining characters parse as a negative integer is appropriately understood.
This allows :pbt :x-10 where :pbt (:MAX -10) or :pbt (:HERE -10) were
previously used.  The old forms are still legal.

The order of the arguments to defcong has been changed.

The simplifier now reports the use of unspecified built-in type information
about the primitives with the phrase ''primitive type reasoning.''  This
phrase may sometimes occur in situations where ''propositional calculus'' was
formerly credited with the proof.

The function pairlis has been replaced in the code by a new function
pairlis$, because Common Lisp does not adequately specify its
pairlis function.

Some new Common Lisp functions have been added, including logtest,
logcount, integer-length, make-list, remove-duplicates, string, and
concatenate.  The source file /slocal/src/acl2/axioms.lisp is the
ultimate reference regarding Common Lisp functions in Acl2.

The functions DEFUNS and THEORY-INVARIANT have been documented.
See :DOC defuns and :DOC theory-invariant.

A few symbols have been added to the list *acl2-exports*.

A new key has been implemented for the acl2-defaults-table,
:irrelevant-formals-ok.  See :DOC set-irrelevant-formals-ok.

The connected book directory, cbd, must be nonempty and begin and end with a
slash.  It is set (and displayed) automatically upon your first entry to LP.
You may change the setting with set-cbd.  See :DOC cbd.

:oops will undo the last :ubt.  See :DOC oops.

Documentation has been written about the ordinals.  See :DOC e0-ordinalp
and :DOC e0-ord-<.

The color events -- (red), (pink), (blue), and (gold) -- may no
longer be enclosed inside calls of LOCAL, for soundness reasons.  In
```

fact, neither may any event that sets the acl2-defaults-table. See :DOC embedded-event-form.

See :DOC ld-keyword-aliases for an example of how to change the exit keyword from :q to something else.

The attempt to install a monitor on :REWRITE rules stored as simple abbreviations now causes an error because the application of abbreviations is not tracked.

A new message is sometimes printed by the theorem prover, indicating that a given simplification is ''specious'' because the subgoals it produces include the input goal. In Version 1.4 this was detected but not reported, causing behavior some users found bizarre. See :DOC specious-simplification.

:DEFINITION rules are no longer always required to specify the :CLIQUE and :CONTROLLER-ALIST fields; those fields can be defaulted to system-determined values in many common instances. See :DOC definition.

A warning is printed if a macro form with keyword arguments is given duplicate keyword values. Execute (thm t :doc nil :doc "Ignored") and read the warning printed.

A new restriction has been placed on ENCAPSULATE. Non-LOCAL recursive definitions inside the ENCAPSULATE may not use, in their tests and recursive calls, the constrained functions introduced by the ENCAPSULATE. See :DOC subversive-inductions.

The events defequiv, defcong, defrefinement, and defevaluator have been reimplemented so that they are just macros that expand into appropriate defthm or encapsulate events; they are no longer primitive events. See the documentation of each affected event.

The defcor event, which was a shorthand for a defthm that established a corollary of a named, previously proved event, has been eliminated because its implementation relied on a technique we have decided to ban from our code. If you want the effect of a defcor in Version 1.5 you must submit the corresponding defthm with a :by hint naming the previously proved event.

Error reporting has been improved for inappropriate in-theory hints and events, and for syntax errors in rule classes, and for non-existent filename arguments to LD.

Technical Note: We now maintain the Third Invariant on type-alists, as described in the Essay on the Invariants on Type-alists, and Canonicality. This change will affect some proofs, for example, by causing a to rewrite more quickly to c when (equiv a b) and (equiv b c) are both known and c is the canonical representative of the three.

VERSION 1.6 RELEASE NOTES

A new key has been implemented for the acl2-defaults-table, :ignore-ok. See :DOC set-ignore-ok.

It is now legal to have color events, such as (RED), in the portcullis of a book. More generally, it is legal to set the acl2-defaults-table in the portcullis of a book. For example, if you execute :RED and then certify a book, the event (RED) will show up in the portcullis of that book, and hence the definitions in that book will all be red (except when overridden by appropriate declarations or events). When that book is included, then as always, its portcullis must first be ''raised,'' and that will cause the default color to become red before the events in the book are executed. As always, the value of acl2-defaults table immediately after execution of an include-book, certify-book, or encapsulate form will be the same as it was immediately before execution (and hence, so will the default color). See :DOC portcullis and, for more about books, :DOC books.

A theory GROUND-ZERO has been defined to contain exactly those rules that are enabled when Acl2 starts up. See :DOC ground-zero.

The function nth is now enabled, correcting an oversight from Version 1.5.

Customization files no longer need to meet the syntactic restrictions put on books; rather, they can contain arbitrary Acl2 forms. See :DOC acl2-customization.

Structured directory names and structured file names are supported; see especially :DOC pathname, :DOC book-name, and :DOC cbd.

Acl2 now works with some Common Lisp implementations other than akcl, including Lucid, Allegro, and MCL.

A facility has been added for displaying proof trees, especially using emacs; see :DOC proof-tree.

There is a considerable amount of new documentation, in particular for the printing functions FMT, FMT1, and FMS, and for the notion of Acl2 term (see :DOC term).

It is possible to introduce new well-founded relations, to specify which relation should be used by DEFUN, and to set a default relation. See :DOC well-founded-relation.

It is possible to make functions suggest new inductions. See :DOC induction.

It is possible to change how Acl2 expresses type-set information; in particular, this affects what clauses are proved when forced assumptions are generated. See :DOC type-set-inverter.

A new restriction has been added to DEFPKG, having to do with undoing. If you undo a DEFPKG and define the same package name again, the imports list must be identical to the previous imports or else an explanatory error will occur. See :DOC package-reincarnation-import-restrictions.

Theory-invariant and set-irrelevant-formals-ok are now embedded event forms.

The command :GOOD-BYE may now be used to quit entirely out of Lisp, thus losing your work forever. This command works in akcl but may not work in every Common Lisp.

A theory GROUND-ZERO has been added that contains exactly the enabled rules in the startup theory. See :DOC ground-zero.

DEFINE-PC-MACRO and DEFINE-PC-ATOMIC-MACRO now automatically define :red functions. (It used to be necessary, in general, to change color to :red before invoking these.)

A proof of the well-foundedness of e0-ord-< on the e0-ordinalps is in :DOC proof-of-well-foundedness.

Free variables are now handled properly for hypotheses of :type-prescription rules.

When the system is loaded or saved, STATE is now bound to *THE-LIVE-STATE*.

Certify-book has been modified so that when it compiles a file, it loads that object file.

Defstub has been modified so that it works when the color is hot (:red or :pink).

Several basic, but not particularly commonly used, events have been added or changed. The obscure axiom SYMBOL-NAME-INTERN has been modified. The definition of FIRSTN has been changed. BUTLAST is now defined. The definition of INTEGER-LENGTH has been modified. The left-hand side of the rewrite rule rational-implies2 has been changed from (* (numerator x) (/ (denominator x))) to (* (/ (denominator x)) (numerator x)), in order to respect the fact that unary-/ is invisible with respect to binary-*. See :DOC loop-stopper.

The 'preprocess' process in the waterfall (see the discussion of the :do-not hint in :DOC hints) has been changed so that it works to avoid case-splitting. The 'simplify' process refuses to force (see :DOC force) when there are IF terms, including AND and OR terms, in the goal being simplified.

The function APPLY is no longer introduced automatically by translation of user input to internal form when functions are called on inappropriate explicit values, e.g., (car 3).

The choice of which variable to use as the measured variable in a recursive definition has been very slightly changed.